Can OWL and Logic Programming Live Together Happily Ever After?

Boris Motik[1], Ian Horrocks[1], Riccardo Rosati[2], and Ulrike Sattler[1]

[1] University of Manchester, Manchester, UK
[2] Università di Roma "La Sapienza", Rome, Italy

Abstract. Logic programming (LP) is often seen as a way to overcome several shortcomings of the Web Ontology Language (OWL), such as the inability to model integrity constraints or perform closed-world querying. However, the open-world semantics of OWL seems to be fundamentally incompatible with the closed-world semantics of LP. This has sparked a heated debate in the Semantic Web community, resulting in proposals for alternative ontology languages based entirely on logic programming. To help resolving this debate, we investigate the practical use cases which seem to be addressed by logic programming. In fact, many of these requirements have already been addressed outside the Semantic Web. By drawing inspiration from these existing formalisms, we present a novel logic of *hybrid MKNF knowledge bases*, which seamlessly integrates OWL with LP. We are thus capable of addressing the identified use cases without a radical change in the architecture of the Semantic Web.

1 Introduction

In the past couple of years, a significant body of Semantic Web research was devoted to defining a suitable language for ontology modeling. In 2004, this endeavor resulted in the Web Ontology Language (OWL). OWL is based on Description Logics (DLs) [**?**]—a family of knowledge representation formalisms based on first-order logic and exhibiting well-understood computational properties. OWL has been successfully applied to numerous problems in computer science, such as information integration or metadata management. Prototypes of OWL reasoners,[3] such as RACER, FaCT++, Pellet, or KAON2, have been implemented and applied in research projects; commercial implementations and projects using them are currently emerging.

However, the experience in building practical applications has revealed several shortcomings of OWL. For example, OWL does not allow for integrity constraints or closed-world reasoning. Rule-based formalisms grounded in logic programming have repeatedly been proposed as a possible solution, so adding a rule layer on top of OWL is nowadays seen as a central task in the development of the Semantic Web language stack. The Rule Interchange Format (RIF) work-

[3] A list of reasoners is available at http://www.cs.man.ac.uk/∼sattler/reasoners.html.

ing group[4] of the World Wide Web Consortium (W3C) is currently working on standardizing such a language.

Responding to popular demand, the Semantic Web Rule Language (SWRL) was proposed in [**?**]. However, as the authors point out, SWRL is a simple extension of OWL with material (first-order) implication and, due to the straightforward way in which the rules are integrated with OWL, it is trivially undecidable. Furthermore, SWRL was designed as a first-order language, so it does not address nonmonotonic reasoning tasks, such as expressing integrity constraints. OWL and SWRL were criticized on these accounts in [**?**], and an alternative ontology language OWL-Flight, based entirely on logic programming, was proposed. In [**?**], the authors go even further by saying that a true rule formalism grounded in logic programming is intrinsically incompatible with OWL. They propose to change the layering architecture of the Semantic Web: instead of building rules on top of OWL, they propose OWL and rules to exist side-by-side, with semantic interoperability grounded in Description Logic Programs (DLP) [**?**]—a straightforward intersection of DLs and LP. Furthermore, the authors propose the Web Service Modeling Language (WSML) [**?**] or F-Logic [**?**] as suitable ontology languages based on logic programming. These approaches were criticized in [**?**] on the grounds that separating OWL and rules creates two Semantic Webs with little or no semantic interoperability.

To help in resolving this debate, in Section 3 we investigate the practical use cases which are difficult or impossible to realize in OWL, but seem to be addressed by logic programming. These use cases are not novel to knowledge representation: numerous formalisms addressing different subsets of these requirements have already been developed, so we present an overview of the most relevant ones in Section 4. Many existing proposals are based on description logics, so analyzing them provides valuable insights into integrating logic programming with OWL without sacrificing backwards compatibility.

By combining the ideas from the existing formalisms with the principles of logic programming, we developed a novel formalism of *hybrid MKNF knowledge bases*, which we overview in Section 5. This formalism, based on the logic MKNF by Lifschitz [**?**], is fully compatible with both OWL and logic programming, and thus addresses the identified use cases without sacrificing backwards compatibility. Because it subsumes logic programming, our logic provides a foundation for integrating OWL with languages such as WSML and F-Logic. Thus, it is possible to obtain a coherent stack of logical languages without establishing the "twin towers of the Semantic Web" [**?**], and our formalism provides a framework for integrating several proposals considered within RIF.

Due to space constraints, we present hybrid MKNF knowledge bases only at a high level by means of an example. For precise definitions and decision procedures, please refer to [**?**].

[4] http://www.w3.org/2005/rules/

2 Preliminaries

2.1 The OWL Family of Languages

OWL is actually a family of three ontology languages: OWL-Lite, OWL-DL, and OWL-Full. The first two languages can be considered syntactic variants of the $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$ description logics, respectively, whereas the third language was designed to provide full compatibility with RDF(S). We focus mainly on the first two variants of OWL because OWL-Full has a nonstandard semantics that makes the language undecidable and therefore difficult to implement. OWL comes with several syntaxes, all of which are rather verbose. Hence, in this paper we use the standard DL syntax, which we overview next. For a full introduction to the syntax and the semantics of DLs, please refer to [**?**].

The main building blocks of DL knowledge bases are *concepts* (or *classes*), representing sets of objects, *roles* (or *properties*), representing relationships between objects, and *individuals*, representing specific objects. Concepts such as *Person* are *atomic*. Using a rich set of concept constructors, one can construct *complex* concepts, which describe the conditions on concept membership. For example, the concept $\exists \mathit{hasFather}.\mathit{Person}$ describes those objects that are related through the *hasFather* role with an object from the concept *Person*. A DL knowledge base $\mathcal{O}$ typically consists of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$. A TBox contains axioms about the general structure of all allowed worlds, and is therefore akin to a database schema. For example, the TBox axiom (1) states that each instance of the concept *Person* must be related by the role *hasFather* with an instance of the concept *Person*. An ABox contains axioms that describe the structure of a particular world. For example, the axiom (2) states that *Peter* is a *Person*, and (3) states that *Paul* is a brother of *Peter*.

$$\mathit{Person} \sqsubseteq \exists \mathit{hasFather}.\mathit{Person} \tag{1}$$

$$\mathit{Person}(\mathit{Peter}) \tag{2}$$

$$\mathit{hasBrother}(\mathit{Peter}, \mathit{Paul}) \tag{3}$$

A DL knowledge base can be given semantics by translating it into first-order logic with equality. Atomic concepts are translated into unary predicates, complex concepts into formulae with one free variable, and roles into binary predicates. The basic reasoning problems for OWL are checking if an individual a is an instance of a concept C (written $\mathcal{O} \models C(a)$) or if the a concept C is subsumed by another concept D (written $\mathcal{O} \models C \sqsubseteq D$). These problems are decidable for OWL-Lite and OWL-DL in ExpTime and NExpTime, respectively.

The concept-centric style of modeling endorsed by OWL has proven to be particularly suitable for modeling taxonomic knowledge. Furthermore, the open-world semantics of OWL grounded in first-order logic allows one to state general truths, and not only statements about known objects. In fact, in OWL one can introduce new, unknown individuals to express such truths, which provides an elegant way of modeling incomplete information.

2.2 Logic Programming

Logic programming (LP) is a family of KR formalisms centered around the notion of rules—statements of the following form:

$$H \leftarrow B_1^+, \ldots, B_n^+, \mathbf{not}\, B_1^-, \ldots, \mathbf{not}\, B_k^- \tag{4}$$

Different semantics for LP have been considered in practice, with stable models [**?**] being the most widely accepted one: a set of atoms M is a *stable model* of a set of rules P if it is the minimal model of a program P^M, where the latter is obtained by replacing each atom $\mathbf{not}\, B_i^-$ with its value in M. A set of rules P can have zero, one, or several stable models, and checking satisfiability of P is an NP-complete problem, assuming P is function-free. Numerous variants of these basic formalisms have been considered, such as rules with disjunctions in the rule heads or extensions with classical negation; a combination of these two features is commonly known as *answer set programming* [**?**].

F-Logic [**?**] is a language layered on top of logic programming, providing object-oriented primitives for modeling concept hierarchies, concept instantiation, relationships between individuals, and inheritance. For execution, F-Logic theories can be compiled into logic programming; hence, the relationship between F-Logic and LP is somewhat similar to the relationship between C++ and assembler. OWL-Flight [**?**] and the Web Service Modeling Language (WSML) [**?**] are other notable object-oriented front-ends for logic programming.

Logic programming partly evolved as an extension of relational databases with deductive features. Therefore, LP typically focuses on efficient query answering over a bounded data set, and is often used in data-intensive applications that require managing large amounts of data. With the introduction of answer set programming, LP is increasingly seen as a general problem-solving formalism, capable of succinctly expressing hard computational problems.

3 Why Integrate OWL With Logic Programming?

In this section we motivate the need for integrating OWL and LP. In particular, we present several important modeling problems that are hard, if not impossible to solve using OWL alone, but can easily be addressed using logic programming.

Higher Relational Expressivity. OWL provides a rich set of primitives for expressing concepts; however, the set of primitives regarding roles is often not sufficient for practical applications. Roughly speaking, OWL can model only domains where objects are connected in a tree-like manner; however, many real-world applications require modeling general relational structures. For example, saying that "an uncle of a person is a brother of that person's father" requires expressing a triangle between the person, the father, and the uncle. An in-depth discussion about the relational expressivity of OWL can be found in [**?**].

Polyadic Predicates. The basic modeling constructs of OWL are concepts and roles, which correspond to unary and binary predicates. However, many relationships encountered in practice are of arity larger than two. For example, flight connections between cities together with the airline providing the service can naturally be represented using a ternary predicate, so $\mathit{flight}(\mathit{MAN}, \mathit{STR}, \mathit{HLX})$ might mean that HLX offers flights between Manchester and Stuttgart.

Closed-World Reasoning. Consider an OWL knowledge base $\mathcal{O}$ containing an assertion $\mathit{flight}(a, b)$ for each pair of cities connected by a flight. Due to the *open-world* semantics of OWL, we can use $\mathcal{O}$ to answer positive queries—that is, queries about which cities are connected by a flight. However, we cannot use $\mathcal{O}$ to answer negative queries: $\mathcal{O}$ does not contain explicit information about not connected cities, so, for each c and d, we have $\mathcal{O} \not\models \neg\mathit{flight}(c, d)$. Answering queries about negative information in an intuitive way usually requires some form of *closed-world* reasoning.

The difference between open- and closed-world reasoning can be intuitively described as follows. In first-order logic, if a fact α holds only in a subset of the models of $\mathcal{O}$, then we can conclude neither $\mathcal{O} \models \alpha$ nor $\mathcal{O} \not\models \alpha$; in a way, $\mathcal{O}$ is *underspecified* with respect to α. In contrast, closed-world formalisms make the common-sense conjecture that all relevant information is explicitly known, so all unprovable facts should be assumed not to hold in $\mathcal{O}$. Hence, closed-world reasoning can be understood as reasoning where $\mathcal{O} \not\models \alpha$ implies $\mathcal{O} \models \neg\alpha$.

The requirement for closed-world reasoning comes in practice in two distinct forms. Certain applications require only *closed-world querying* of open-world knowledge bases. A closed-world query language can be layered on top of OWL without changing the semantics of OWL itself.

Alternatively, closed-world reasoning can be integrated into the reasoning process itself. For example, after determining that c and d are not connected by a flight, a travel planning application might check for a train connection. This is usually enabled through a form of *default* or *weak* negation, commonly denoted with **not**. Default negation is closely related to closed-world reasoning: intuitively, from $\mathcal{O} \not\models \alpha$ one concludes $\mathcal{O} \models \mathbf{not}\, \alpha$. Unlike a closed-world query language, default negation must be built into the foundations of the knowledge representation formalism, affecting its semantics significantly.

We point out two common misconceptions about closed-world reasoning. The first one is that closed-world reasoning can be emulated within first-order logic by specifying complete information—for example, using a form of *role closure*. The following axiom states that flights exist only between cities a and b, and b and c, thus making the role *flight* closed:

$$\forall x, y : \mathit{flight}(x, y) \leftrightarrow (x \approx a \wedge y \approx b) \vee (x \approx b \wedge y \approx c) \tag{5}$$

Assuming that $\mathcal{O}$ contains only (5), we can now conclude $\mathcal{O} \models \neg\mathit{flight}(a, d)$, so role closure seems to solve the problem. However, such a solution is not satisfactory since it does not provide the required support for inferencing. For example, it is natural to query $\mathcal{O}$ for nondirect flights between cities—that is,

to query the transitive closure of *flight*. A natural solution is to add a transitive role *anyLengthFlight* and the axiom $\mathit{flight} \sqsubseteq \mathit{anyLengthFlight}$. However, $\mathcal{O}$ can again answer only positive queries, since we did not say that *anyLengthFlight* is a *minimal* transitive relation containing *flight*. In fact, transitive closure is not axiomatizable in first-order logic, so answering our (quite natural) query requires some form of closed-world, non-first-order reasoning.

Furthermore, closed-world reasoning is often confused with closed-domain reasoning. Consider a knowledge base $\mathcal{O}$ containing axioms (1)–(3). Axioms (1) and (2) state that *Peter* has a father without saying who the father is. The only persons known in $\mathcal{O}$ are *Peter* and *Paul*, but in open-domain reasoning the unnamed father of *Peter* is not required to be either of them: the existential quantifier in (1) can refer to an object not explicitly mentioned by name. However, the fact that the existential quantifier makes the *domain* of the ontology open is unrelated to the problems of open- or closed-world reasoning. As explained earlier, closed-world reasoning is about drawing common-sense conjectures regarding explicit or implicit objects; it has nothing to do with the ability to refer to new individuals. In fact, closing the domain of $\mathcal{O}$ can be done without leaving first-order logic, by including the axiom $\top \sqsubseteq \{\mathit{Peter}, \mathit{Paul}\}$. Now, the father of *Peter* is either *Paul* or *Peter* (note that we did not say that fatherhood is acyclic), so the domain of $\mathcal{O}$ is closed. However, closing the domain does not provide any new default consequences. For example, the sex of *Peter* has not been explicitly specified, so $\mathcal{O} \not\models \mathit{Man}(\mathit{Peter})$ and $\mathcal{O} \not\models \neg \mathit{Man}(\mathit{Peter})$. It is also possible to combine closed-world reasoning with the ability to refer to unknown individuals. For example, if we additionally state that *Peter*, *Paul*, and the unnamed father are different objects, by closed-world reasoning we can deduce that the domain contains exactly three objects, even though only two individuals are known by name. The domain in this example is open in the (weaker) sense that it is not restricted to named individuals.

Integrity Constraints. In OWL, domain and range restrictions constrain the type of objects that can be related by a role. For example, (6) states that fatherhood is defined only for persons and animals. Also, participation restrictions specify that certain objects have relationships to other objects. For example, (7) states that each person has a social security number.

$$\exists \mathit{hasFather}.\top \sqsubseteq \mathit{Person} \sqcup \mathit{Animal} \tag{6}$$

$$\mathit{Person} \sqsubseteq \exists \mathit{hasSSN}.\mathit{SSN} \tag{7}$$

Under standard first-order semantics, (6) and (7) imply new facts: from $\mathcal{O} = \{\mathit{hasFather}(\mathit{Peter}, \mathit{Paul}),\ \mathit{Person}(\mathit{Ann})\}$, we conclude $\mathit{Person} \sqcup \mathit{Animal}(\mathit{Peter})$ and that *Ann* has a social security number (we do not know which one).

Axioms (6) and (7) describe the structure of the world being modeled. However, one often wants to describe the required structure of the knowledge base. In traditional object-oriented modeling, (6) means "fatherhood can be stated only for objects known to be persons or animals"; similarly, (7) means "a social security number must be known for each person." Under such an interpretation, the

axioms (6) and (7) would be interpreted as *integrity constraints*. Now $\mathcal{O}$ would invalidate the integrity constraints, since it is incomplete. It is well-known that integrity constraints cannot be realized within first-order logic [**?**].

Modeling Exceptions. Exceptions abound in the natural world. For example, most people have the heart on the left, but some people (called dextrocardiacs) have it on the right side of the body. Such a domain cannot be modeled in OWL: the axioms $\mathit{Human} \sqsubseteq \mathit{HeartOnLeft}$, $\mathit{Dextrocardiac} \sqsubseteq \mathit{Human}$, and $\mathit{Dextrocardiac} \sqsubseteq \neg \mathit{HeartOnLeft}$ make the concept *Dextrocardiac* unsatisfiable. To enable exception modeling, one must go beyond first-order logic and apply a nonmonotonic formalism, usually involving some form of default negation.

One might argue that exceptions should be handled extralogically: one could preprocess a knowledge base and add an assertion $\mathit{HeartOnLeft}(\alpha)$ to each object α that is provably a *Human* and not an *Dextrocardiac*. However, this solution is far from ideal. The preprocessing algorithm would be defined in an ad-hoc way, thus destroying the well-defined semantics—something deemed to be a crucial feature of OWL. Also, it would be difficult to describe the interaction between preprocessing and the actual reasoning. Nonmonotonic formalisms provide a coherent framework for studying such issues.

4 Existing Solutions to the Problems Mentioned

The use cases from Section 3 are not novel to knowledge representation, and they have been addressed previously by different formalisms. Many of them are based on DLs, so they provide important guidelines for integrating OWL with logic programming without introducing backwards incompatibility.

4.1 First-Order Rule Formalisms for DLs

Many different proposals exist for extending DLs with first-order rules.[5] The general idea is quite simple: one allows for the axioms of the form $H \leftarrow B_1, \ldots, B_n$ where H (the rule *head*) and B_i (the rule *body*) can be of the form $C(s)$ or $R(s,t)$, for C a concept, R a role, and s and t terms (i.e., variables or individuals). The rules are interpreted under standard first-order semantics as $\forall \mathbf{x} : H \vee \neg B_1 \vee \ldots \vee \neg B_n$, where $\mathbf{x}$ is the set of free variables of all H and B_i. The Semantic Web Rule Language (SWRL) [**?**] was layered on top of OWL based on these principles. The following rule models the relationship about uncles from Section 3:

$$\mathit{hasUncle}(x,z) \leftarrow \mathit{hasFather}(x,y), \mathit{hasBrother}(y,z) \tag{8}$$

It is straightforward to extended SWRL with n-ary predicates, in which case one usually distinguishes the *DL-predicates* (the predicates allowed to occur in DL axioms) from the *non-DL-predicates* (the predicates occurring solely in rules).

[5] Some authors insist on calling first-order rules *clauses*, reserving the term "rules" for nonmonotonic formalisms. This has not established itself in the Semantic Web.

First-order extensions of DLs with rules are quite straightforward from the standpoint of the semantics: the rules are actually standard first-order material implications, just like standard DL inclusion axioms. All first-order properties, such as contrapositive inferences, apply to the rules as well: from $A(x) \leftarrow B(x)$ and $\neg A(a)$ we can derive $\neg B(a)$.

Unfortunately, extending DLs with rules significantly affects the computational properties of the resulting formalism. In [**?**], it was shown that integrating recursive Horn rules with even moderately expressive DLs makes reasoning undecidable. Hence, various syntactic restrictions on the rules and the DL have been investigated to regain decidability. For example, CARIN [**?**] proposes *role safety*, according to which at least one variable from a literal with a role predicate must also occur in a non-DL-literal in the rule body. $\mathcal{AL}$-log [**?**] and DL-safe rules [**?**] explore a related notion, which was recently generalized in $\mathcal{DL}$+log [**?**] to *weak safety*: each variable from the *rule head* must occur in a non-DL-literal in the rule body. Weakly safe rules can derive facts only about explicitly known individuals; however, in contrast to $\mathcal{AL}$-log and DL-safe rules, the body literals of $\mathcal{DL}$+log rules can be matched to existentially introduced individuals. Thus, $\mathcal{DL}$+log generalizes conjunctive queries over DL knowledge bases.

Note that the shortcomings in relational expressivity have been partially addressed in the DL $\mathcal{SROIQ}$ [**?**]. This logic extends OWL-DL with *complex role inclusion* axioms, such as $\mathit{hasFather} \circ \mathit{hasBrother} \sqsubseteq \mathit{hasUncle}$, where $\circ$ stands for role concatenation. To make reasoning decidable, these axioms must be *regular*—that is, compatible with a certain acyclic ordering. For example, the previous axiom alone is allowed, but it cannot be used together with the axiom $\mathit{hasChild} \circ \mathit{hasUncle} \sqsubseteq \mathit{hasBrother}$, as this would create a cycle in the definitions of *hasBrother* and *hasUncle*. We discuss the relationship between $\mathcal{SROIQ}$ and rule-based solutions on an example in Section 5.

4.2 Autoepistemic Nonmonotonic Extensions of DLs

Many extensions of DLs with nonmonotonic features are based on autoepistemic logics, as they allow for *introspection*—the ability to reason about one's own beliefs. In these proposals, DLs are extended with an *autoepistemic knowledge operator* $\mathbf{K}$, which can be applied to concepts and roles with an intuitive meaning "is known to hold." Consider again the example from Section 3 of asking whether two cities are not connected by a flight: whereas $\mathcal{O} \not\models \neg \mathit{flight}(c, d)$ holds due to the open-world semantics of OWL, we have $\mathcal{O} \models \neg\, \mathbf{K}\, \mathit{flight}(c, d)$, intuitively meaning that "c and d are not *known* to be connected by a flight." A formula $\mathbf{K}\,\alpha$ is true if α is true in each first-order model I of a knowledge base $\mathcal{O}$. Autoepistemic reasoning can be integrated with DLs in two distinct ways.

Epistemic Operators in Queries. An approach to autoepistemic querying of DL knowledge bases[6] was presented in [**?**], and it was recently generalized to the

[6] Actually, a more general KR formalism was presented in [**?**], in which $\mathbf{K}$ can also occur in the DL knowledge base. However, a reasoning algorithm has been presented only for the case of ordinary knowledge bases and epistemic queries.

Epistemic Query Language (EQL) [**?**]. We overview here EQL-Lite($\mathcal{Q}$)—a fragment of EQL with favorable computational properties. Given a first-order DL query language $\mathcal{Q}$, EQL-Lite($\mathcal{Q}$) queries are first-order formulae built over the atoms of the form $\mathbf{K}\,q$, where q is a query expressed in the language $\mathcal{Q}$. For example, the cities not connected by a flight can be retrieved using the query $q[x, y] = \neg\,\mathbf{K}\,q'[x, y]$, where $q'[x, y] = \mathit{hasFlight}(x, y)$ is a first-order (conjunctive) query. Since $\mathbf{K}$ can occur only in queries, the semantics of $\mathbf{K}$ is layered on top of the standard DL semantics in a nonintrusive way. In fact, $\mathbf{K}$ can be understood as the *consequence* operator, and EQL-Lite($\mathcal{Q}$) can be understood as an algebra for manipulating first-order consequences of $\mathcal{O}$.

Epistemic Operators in the Knowledge Base. Autoepistemic query languages do not provide default negation or exception modeling; to enable such features, autoepistemic reasoning must be tightly integrated with ordinary DL reasoning. This can be achieved by allowing $\mathbf{K}$ to occur in DL axioms. Usually, a negation-as-failure operator **not**—intuitively understood as "can be false"—is added as well. The first-order version of such a logic is known as the logic of minimal knowledge and negation-as-failure (MKNF) [**?**], and it generalizes several important nonmonotonic formalisms, such as logic programming under stable model semantics [**?**] and default logic [**?**].

Based on these principles, the authors extend in [**?**] the DL $\mathcal{ALC}$ with an autoepistemic knowledge operator $\mathbf{K}$ and an autoepistemic assumption operator $\mathbf{A}$ (which is semantically equivalent to $\neg\,\mathbf{not}$ from the first-order MKNF). The authors also present a decision procedure for an expressive fragment of this logic. Such a logic elegantly addresses the problems from Section 3 related to nonmonotonic reasoning, while being fully compatible with the underlying semantics of DLs. It clearly provides for closed-world querying, and $\mathbf{A}$ directly corresponds to default negation. It also enables defining integrity constraints and provides for exception modeling. For example, the problem of dextrocardiacs from Section 3 can be modeled as $\mathbf{K}\,\mathit{Human} \sqcap \neg\,\mathbf{A}\,\mathit{Dextrocardiac} \sqsubseteq \mathbf{K}\,\mathit{HeartOnLeft}$.

5 Integrating OWL and LP by Hybrid MKNF KBs

Related approaches presented in Section 4 may give us important clues on how to seamlessly integrate OWL with logic programming. On the one hand, a rule formalism layered on top of a DL may address the problems related to relational expressivity and the lack of polyadic predicates. On the other hand, the autoepistemic extensions of DLs integrate closed- and open-world reasoning and thus provide a common logical framework for nonmonotonic extensions of DLs. By integrating these two formalisms, we have developed a novel approach that can be used to seamlessly integrate any DL with LP-style rules. Due to space constraints, we give here only a high-level overview of our proposal; for a complete definition, complexity, and decision algorithms, please see [**?**].

A *hybrid MKNF knowledge base* $\mathcal{K}$ consists of a knowledge base $\mathcal{O}$ in any decidable description logic $\mathcal{DL}$ and a set $\mathcal{P}$ of MKNF rules of the following form:

$$\mathbf{K}\, H_1 \vee \ldots \vee \mathbf{K}\, H_n \leftarrow \mathbf{K}\, B_1^+, \ldots, \mathbf{K}\, B_m^+, \mathbf{not}\, B_1^-, \ldots, \mathbf{not}\, B_k^- \tag{9}$$

As in SWRL, H_i, B_i^+, and B_i^- are first-order atoms of the form $P(t_1, \ldots, t_n)$. For $P = \approx$ or a predicate occurring in $\mathcal{O}$, the atom is a DL-atom; otherwise, it is a non-DL-atom. We assume that $\mathcal{DL}$ comes with an operator π that translates any DL knowledge base $\mathcal{O}$ into a formula $\pi(\mathcal{O})$ of first-order logic with equality. The semantics of our formalism is defined by mapping $\mathcal{K}$ into the following first-order MKNF formula,where $\mathbf{x}$ is the set of free variables of a rule r:

$$\pi(\mathcal{K}) = \mathbf{K}\, \pi(\mathcal{O}) \wedge \bigwedge_{r \in \mathcal{P}} \forall \mathbf{x} : r$$

To obtain a logic with intuitive consequences, we make the *standard names assumption*, which imposes certain restrictions on the models of $\pi(\mathcal{K})$; for more details, please refer to [**?**]. All inference problems for $\mathcal{K}$, such as satisfiability or entailment, are defined w.r.t. $\pi(\mathcal{K})$ in the obvious way.

As we discuss in [**?**], such a formalism can fully capture the semantics of SWRL and $\mathcal{DL}$+log [**?**]. The only approach for combining (possibly nonmonotonic) rules with DLs that we are aware of and that cannot be captured using hybrid MKNF rules is the one by Eiter et al. [**?**]

Similarly to related extensions of DLs with rules, our logic is undecidable in the general case. We address this using the well-known concept of DL-safety: an MKNF rule is DL-safe if each variable in the rule occurs in a non-DL-atom of the form $\mathbf{K}\, A$ in the rule body. Notice that a rule r can automatically be made DL-safe by appending to its body a special literal $\mathbf{K}\, O(x)$ for each variable x, and by adding an assertion $\mathbf{K}\, O(\alpha)$ for each individual α occurring in $\mathcal{K}$. We shall discuss the consequences of this transformation on the semantics of r shortly; moreover, an in-depth discussion of this issue can be found in [**?**]. We believe that our approach can easily be extended to handle weakly safe rules.

In [**?**] we present decision procedures for different types of rules. Furthermore, we analyze the data complexity of reasoning (the complexity under the assumption that the TBox and the rules are fixed, but the ABox varies). Assuming that reasoning in $\mathcal{DL}$ is data complete for NP (which is the case for expressive DLs such as $\mathcal{SHIQ}$), our logic has the same complexity as the corresponding fragment of logic programming. Furthermore, we identify fragments with polynomial data complexity, which are particularly interesting for practice.

The semantics of hybrid MKNF knowledge bases exhibits two important properties. On the one hand, it is fully compatible with OWL: if $\mathcal{P} = \emptyset$, then $\mathcal{K} \models \alpha$ if and only if $\mathcal{O} \models \alpha$ for any first-order formula α. In other words, all standard DL questions are answered in the usual way. On the other hand, MKNF is also fully compatible with logic programming: in [**?**] it was shown that a disjunctive logic program under stable model semantics is equivalent to the MKNF theory where each rule is replaced by an MKNF implication (9). Hence, our formalism reduces to logic programming for $\mathcal{O} = \emptyset$. Function symbols are not

Table 1. A Hybrid MKNF Knowledge Base about Cities

(10) $\mathit{historicCity} \sqsubseteq \exists \mathit{hasChurch}.\mathit{church}$	Historic cities have churches.
(11) $\mathit{church} \sqsubseteq \exists \mathit{designedBy}.\mathit{architect}$	Churches are designed by architects.
(12) $\mathbf{K}\,\mathit{famousCitizen}(x,z) \leftarrow \mathbf{K}\,\mathit{hasChurch}(x,y), \mathbf{K}\,\mathit{designedBy}(y,z), \mathbf{K}\,O(x), \mathbf{K}\,O(y), \mathbf{K}\,O(z)$	Architects are famous citizens in cities where they build their churches.
(13) $\exists \mathit{famousCitizen}.\top \sqsubseteq \mathit{interestingCity}$	Cities with famous people are interesting.
(14) $\mathit{historicCity}(\mathit{Barcelona})$	Barcelona is a historic city.
(15) $\mathit{hasChurch}(\mathit{Barcelona}, \mathit{SagradaFamilia})$	The famous church in Barcelona...
(16) $\mathit{designedBy}(\mathit{SagradaFamilia}, \mathit{Gaudi})$	...was designed by Antonio Gaudi.
(17) $\mathit{seasideCity} \sqsubseteq \exists \mathit{hasRegion}.\mathit{beach}$	Seaside cities have a beach.
(18) $\mathit{beach} \sqsubseteq \mathit{recreational}$	Beaches are for recreation.
(19) $\exists \mathit{hasRegion}.\mathit{recreational} \equiv \mathit{livableCity}$	Livable cities provide for recreation.
(20) $\mathit{portCity}(\mathit{Barcelona})$	Barcelona is a city with a port.
(21) $\mathit{portCity}(\mathit{Hamburg})$	Hamburg is a city with a port.
(22) $\neg \mathit{seasideCity}(\mathit{Hamburg})$	Hamburg is not a seaside city.
(23) $\mathbf{K}\,\mathit{DesignOK}(x) \leftarrow \mathbf{K}\,\mathit{designedBy}(x,y), \mathbf{K}\,O(x), \mathbf{K}\,O(y)$	Auxiliary for the following rule.
(24) $\leftarrow \mathbf{K}\,\mathit{church}(x), \mathbf{not}\ \mathit{DesignOK}(x), \mathbf{K}\,O(x)$	Each church must have an architect.
(25) $\mathit{church}(\mathit{HolyFamily})$	Holy Family is a church.
(26) $\mathit{HolyFamily} \approx \mathit{SagradaFamilia}$	Definition of synonyms.
(27) $\neg \mathit{seasideCity} \equiv \mathit{notSC}$	An atomic name for $\neg \mathit{seasideCity}$.
(28) $\mathbf{K}\,\mathit{seasideCity}(x) \leftarrow \mathbf{K}\,\mathit{portCity}(x), \mathbf{not}\ \mathit{notSC}(x), \mathbf{K}\,O(x)$	Port cities are usually at the seaside.
(29) $\mathbf{K}\,\mathit{Suggest}(x) \leftarrow \mathbf{K}\,\mathit{livableCity}(x), \mathbf{K}\,\mathit{historicCity}(x)$	Suggest to visit livable and historic cities.
(30) $\neg \mathit{livableCity} \equiv \mathit{notLivableCity}$	An atomic name for $\neg \mathit{livable}$.
(31) $\mathbf{K}\,\mathit{Consider}(x) \leftarrow \mathbf{not}\ \mathit{notLivableCity}(x), \mathbf{K}\,O(x)$	Take cities that are not known to be unlivable into consideration as well.

Note: DL-predicates start with a lowercase, and non-DL-predicates with an uppercase letter. There is an assertion $O(\alpha)$ for each object α.

allowed to occur in the rules, since this would make query answering undecidable. However, from the standpoint of the semantics, extending the formalism with function symbols is straightforward, and identifying decidable fragments is an interesting topic for future research. Finally, MKNF rules can also be used to integrate OWL with languages providing an object-oriented view over logic programming, such as F-Logic or WSML.

At first glance, our proposal may seem to be difficult to use and understand. However, we believe MKNF rules to be quite intuitive: just read $\mathbf{K}\,A$ as "A is known to hold" and $\mathbf{not}\ A$ as "it is possible for A not to hold." We demonstrate this on the following example, which also shows how MKNF rules address the requirements from Section 3. Imagine a system helping us to decide where to go on holiday, based on the tourism ontology $\mathcal{K}$ shown in Table 1.

The impact of DL-safety is demonstrated by axioms (10)–(13). By (10) and (11), each historic city α has at least one church β, which has at least one architect γ. By (12), γ is a famous citizen of α so, by (13), α is an interesting city. Now if (12) were a normal (non-DL-safe, first-order) rule, one might perform this inference for *any* individuals α, β, and γ, which would thus imply $\mathcal{K} \models \mathit{historicCity} \sqsubseteq \mathit{interestingCity}$. However, (12) is DL-safe—all variables occur in an atom with the predicate O. Hence, it is applicable only to the individ-

uals *known in the ABox by name*, and not to those introduced by the existential quantifier, so we cannot conclude that *interestingCity* subsumes *historicCity*. Note that (12) could be stated in $\mathcal{SROIQ}$ using a "non-DL-safe" role inclusion axiom, and this would correctly imply the subsumption relationship.

Whereas making rules DL-safe usually restricts the subsumption inferences, it typically has less impact on ABox query answering. Namely, (14)–(16) specify the names of a church in Barcelona and its architect. All variables in (12) can now be bound to known individuals, so $\mathcal{K} \models \mathit{famousCitizen}(\mathit{Barcelona}, \mathit{Gaudi})$; by (13), we derive $\mathcal{K} \models \mathit{interestingCity}(\mathit{Barcelona})$. Hence, DL-safety is a compromise that provides for ABox query answering at the expense of some subsumption inferences if expressivity beyond $\mathcal{SROIQ}$ is needed, but without losing decidability. DL-safety is crucial for nonmonotonic reasoning: without it, most nonmonotonic logics with existential quantification are not even semidecidable.

Consider an integrity constraint requiring that an architect should be explicitly specified for each explicitly mentioned church. One might intuitively write the rule $\leftarrow \mathbf{K}\, \mathit{church}(x), \mathbf{not}\ \mathit{designedBy}(x, y), \mathbf{K}\, O(x), \mathbf{K}\, O(y)$ (paraphrased as "it is an error to have a known church without a known designer"). However, this rule is incorrect: all variables in rules are universally quantified, so this rule requires each church to be connected through *designedBy* to each other object. To formulate the integrity constraint correctly, we introduce the auxiliary rule (23) which projects the variable y from $\mathit{designedBy}(x, y)$, and then use the result in (24) to identify the churches without a designer.

Nonmonotonic formalisms usually assume that distinct constants mean different things—a feature known as *unique name assumption* (UNA). Let us for the moment assume that $\mathcal{K}$ does not contain (26). We would then intuitively expect (24) to be violated, since the designer of *HolyFamily* has not been specified. However, without UNA, $\mathcal{K}$ would be satisfiable, and it would entail that *HolyFamily* and *SagradaFamilia* are the same things. To avoid such counterintuitive consequences, logic programming assumes UNA by default.

In contrast, OWL does not employ UNA: explicit equality statements can be used to define synonyms. We integrate OWL with logic programming by using the standard names assumption. Roughly speaking, we allow that two individuals are equal only if there is explicit evidence for doing so. For more information on this issue, please refer to [**?**]; we just note here that such a semantics does not change any standard OWL consequences. Returning to our example, we make *HolyFamily* and *SagradaFamilia* synonyms by (26), which then makes (24) satisfied for (16) and (25).

Rule (28) asserts the common-sense knowledge that port cities are usually at the seaside, allowing us to conclude $\mathcal{K} \models \mathit{seasideCity}(\mathit{Barcelona})$. However, (28) allows for exceptions: the atom $\mathbf{not}\ \mathit{notSC}(x)$ basically says "if not proven not to be at the seaside." (Axiom (27) is needed because only atomic concepts can occur in MKNF rules.) According to (22), Hamburg is an exception (it is located on the river Elbe), so the default conclusion $\mathcal{K} \models \mathit{seasideCity}(\mathit{Hamburg})$ of (28) is suppressed, as it would lead to contradiction.

The rule (29) is intended as a query that suggests which cities to visit. Even though the conclusion $seasideCity(Barcelona)$ was derived by nonmonotonic reasoning, it implies further conclusions through monotonic reasoning. Namely, axioms (17)–(19) imply $\mathcal{K} \models livableCity(Barcelona)$, which is derived by standard DL reasoning involving unnamed individuals (introduced by $\exists hasRegion.Beach$). Hence, $\mathcal{K} \models Suggest(Barcelona)$.

Finally, (31) shows how default negation is layered over open-world semantics. Intuitively, MKNF performs open- and closed-world inferences "in parallel." For example, $\mathcal{K} \not\models livableCity(Hamburg)$ and $\mathcal{K} \not\models \neg livableCity(Hamburg)$ hold according to the usual DL semantics. By reformulating these questions with closed-world interpretation in mind, we get $\mathcal{K} \models \mathbf{not}\ livableCity(Hamburg)$ (Hamburg is not known to be livable) and $\mathcal{K} \models \mathbf{not}\ notLivable(Hamburg)$ (Hamburg is not known not to be livable either). Hence, (31) allows us to conclude $Consider(Hamburg)$—even though we do not know for sure that Hamburg is a livable city, we do not know the opposite either, so it might still be worth a visit. Intuitively speaking, the DL part of $\mathcal{K}$ is interpreted under open-world semantics; however, **K** and **not** allow the user to put on "closed-world glasses" and examine the nonmonotonic consequences of the DL part. By using these consequences in rules, one can enforce new nonmonotonic conclusions.

6 Conclusion

Motivated by the ongoing controversy in the Semantic Web community about the proper layering of a nonmonotonic rule formalism on top of OWL, we analyze the shortcomings of OWL that are deemed to be solvable using logic programming. Furthermore, we overview existing formalisms that address these requirements. We thus gain insight into how OWL could be integrated with rules without sacrificing semantic compatibility with either formalism.

By combining the ideas of SWRL and DL-safe rules with the approaches for autoepistemic extensions of DLs, we propose a new formalism of hybrid MKNF knowledge bases that seamlessly integrates OWL with logic programming. We present the features of our formalism on a nontrivial example. Under the standard DL-safety assumption, our formalism is decidable, and its data complexity is not higher than for plain logic programming. Therefore, our formalism provides a solid foundation for the integration of OWL and logic programming, as well as a framework for integrating several considered proposals within RIF.

The main challenge for our future work is to implement our approach in the KAON2[7] reasoner and thus validate the usefulness of our formalism in practice.

[7] http://kaon2.semanticweb.org/